CHAPTER 52

SOFTWARE RELIABILITY EVALUATION

CONTENTS

1 INTRODUCTION

1.1 This chapter describes the methods that can be adopted to evaluate the software reliability that has actually been achieved, including evidence from testing, field data, fault data and analytical arguments. More information on the methods and techniques discussed in this chapter can be obtained from publications listed in leaflet 52/0.

1.2 Where possible, software reliability evaluation should be carried out as part of normal system reliability tests and trials, to minimize costs and to exercise the software in the system environment.

2 EVIDENCE FROM TESTING

2.1 Reliability Growth Modeling

2.1.1 This technique attempts to predict in-service reliability from data on the times between failures as the software is tested and design faults removed towards the end of the development phase. It can be used to assess the effectiveness of the software engineering process, and to predict when the software will meet its reliability requirements. Various software reliability growth models have been developed and there is software available to perform the extensive calculations that are required. See BS 5760 Part 8[1] for further details.

2.1.2 While reliability growth modeling makes it possible in principle to obtain measures of the operational reliability of a program, there are several difficulties with using the approach on software with stringent reliability requirements. Most importantly, there is a strong law of diminishing returns operating when software is debugged by waiting for faults to be revealed during testing.

2.1.3 Reliability growth modeling is based on testing during development. The expectation is that the software will meet its reliability requirements in service, but sometimes the exposure of many copies of the software to operational conditions will reveal additional failures. Further reliability growth can occur if arrangements have been made so that design faults that lead to in-service failures are diagnosed and corrected (see also PtCCh51 paragraph 2.19).

2.2 Statistical Testing

2.2.1 The demonstration of achieved software reliability through tests and trials should be a key part of the software reliability programme. The aim should be to subject the software to test data that is statistically similar to the in-service environment. Tests can either be carried out with the software installed in the system, or by means of a simulator.

2.2.2 System reliability growth testing, as discussed in PtCCh15 will provide reliability data under operational conditions.

2.2.3 The tests should be carried out over a period several times the required mean time to failure. It may be difficult to attain this with software with very stringent reliability requirements. Some ways in which testing can be speeded up are:

a) Testing many copies of the software in parallel with different data sets;

b) Testing on a faster computer than the eventual target using identical code; and

c) Using goal-directed ("trajectory") testing. For a system that is only required to operate occasionally, such as a shut-down system, the test data can concentrate on values that should initiate action. Because of memory effects, a period of working prior to the demand should be included.

2.2.4 Because of the large numbers of tests involved, it is desirable to use a diverse implementation of the software to check the results. This is known as "back-to-back" testing. Prototype versions or a specification animation (see PtCCh51 paragraph 2.10) can be used as the diverse implementation. Alternatively, the tests can be checked against a real-world simulator to establish that failures do not occur.

2.2.5 Very high reliability software may not fail at all during statistical testing. If a particular version of the software has not failed on test, and if the testing has been carried out for a time t, it can be shown that there is a 50:50 chance that the software will run for another time *t* without failing. See Leaflet 52/0 (5) for details of this result, and Leaflet 52/0 (8) for information on stopping rules for testing.

2.3 Performance Testing

2.3.1 Testing may be carried out to establish that the non-functional performance requirements have been met. Such tests should include:

a) Tests of specific design features for reliability (e.g. overload handling and interface error handling);

b) Overload/stress tests (see PtCCh51 paragraph 2.17.7);

c) Fault tolerance tests, including tests of fault detection, recovery and isolation mechanisms (see PtCCh51 paragraph 3). This may be difficult to test in at system level and so tests may have to be carried out on individual components separately;

d) Normal and worst case resource utilisation measurements; and

e) Real-time performance tests (e.g. time response and throughput).

3 USE OF FIELD DATA

3.1 The use of previous operational history of software that is to be reused may provide a means of evaluating software reliability. Caution is needed, however, since manual reporting schemes suffer from under-reporting and misdiagnosis, especially if the failure is transient or the system is repaired in the field. More accurate failure data may be obtainable through the use of BIT (see Def Stan 00-42 part 4[2]).

3.2 The use of operational history is a valid approach provided that the particular software product has been used in a range of different applications that give good coverage of the most typical parts of the input space. This approach has to be complemented by good configuration and quality management systems so that users' problems are recorded.

Unfortunately very few commercial software Contractors will provide failure data to potential users.

4 EVIDENCE FROM FAULT DATA

4.1 Recent research has developed a theory for software reliability growth that allows a conservative bound to be placed on the mean time to failure (MTTF) of software after a period of use on the basis of the initial number of faults. The theory is based on the assumption that a stable operational environment will result in a fixed but unknown failure rate for each defect. The theory shows that the software reliability after a usage time T is bounded by:

$$\text{MTTF} = eT/N \text{ (e times T divided by N)}$$

where *N* is the initial number of faults and e is the exponential constant (2.718.. .).

4.2 The theory assumes that all failures are perfectly diagnosed and the underlying faults are corrected, but the predictions are relatively insensitive to assumption violations over the longer term. Less conservative results can be obtained if additional assumptions are made about the failure rate distribution of faults.

5 EXHAUSTIVE TESTING

5.1 For certain small, simple systems, it may be possible to test all combinations of inputs and internal states. Exhaustive testing needs to be carried out with special care for software since changes in the internal state over time may cause subtle errors (for example, the memory may overflow after a large number of cycles if re-initialisation is not properly performed). Exhaustive testing is in effect a form of analytical argument because it provides assurance that the software is logically correct with respect to its specification, and has the advantage of including the compiler, link-loader and, if carried out on the target system, the processor in the testing. However, it will not be practicable for the great majority of systems.

6 RELIABILITY OF SOFTWARE

6.1 Trying to quantify the reliability and quality of software is a difficult and time consuming task. Of the methods through which this can be achieved the most cost effective solution, by far, is to ensure that a rigorous software development process is in place and adhered to throughout the life of the product concerned.

There are many models and methods available, some are free, some are expensive, some are very basic, some are complicated. Unfortunately, there is no “one answer fits all” solution available. Therefore, the following provides a list of possible ways forward.

The best method will probably arise from a mixture of the available resources.

It is important that software is addressed adequately within a programme. It may well, in some cases, be an integral part of the equipment or system being purchased however, there will be other times when the software may have to be treated as an element, or component, of the equipment or system being purchased and, as such, will need to have its own set of requirements including such things as reliability requirements and failure definitions detailed within the Systems Requirement Document. This may lead to other things such as an explicit entry within documents such as any Pre-Qualification Questionnaire or the Invitation to Tender and Statement of Work.

Whilst, as previously stated, there are other models and, indeed, standards available, this leaflet aims to set out what we, along with other NATO members, believe to be best practice.

It should be noted that 9126-1 has been superseded by ISO/IEC 25010 (2011) although parts 2,3 and 4 still reference it. This issue will be addressed when ISO updates their information.

ISO/IEC 25010:2011

A quality in use model composed of five characteristics (some of which are further subdivided into subcharacteristics) that relate to the outcome of interaction when a product is used in a particular context of use. This system model is applicable to the complete human-computer system, including both computer systems in use and software products in use.

A product quality model composed of eight characteristics (which are further subdivided into subcharacteristics) that relate to static properties of software and dynamic properties of the computer system. The model is applicable to both computer systems and software products.

The characteristics defined by both models are relevant to all software products and computer systems. The characteristics and subcharacteristics provide consistent terminology for specifying, measuring and evaluating system and software product quality. They also provide a set of quality characteristics against which stated quality requirements can be compared for completeness.

Although the scope of the product quality model is intended to be software and computer systems, many of the characteristics are also relevant to wider systems and services.

ISO/IEC 25012 contains a model for data quality that is complementary to this model.

The scope of the models excludes purely functional properties, but it does include functional suitability.

The scope of application of the quality models includes supporting specification and evaluation of software and software-intensive computer systems from different perspectives by those associated with their acquisition, requirements, development, use, evaluation, support, maintenance, quality assurance and control, and audit. The models can, for example, be used by developers, acquirers, quality assurance and control staff and independent evaluators, particularly those responsible for specifying and evaluating software product quality. Activities during product development that can benefit from the use of the quality models include:

- identifying software and system requirements;
- validating the comprehensiveness of a requirements definition;
- identifying software and system design objectives;
- identifying software and system testing objectives;
- identifying quality control criteria as part of quality assurance;
- identifying acceptance criteria for a software product and/or software-intensive computer system;
- establishing measures of quality characteristics in support of these activities.

ISO/IEC TR 9126-2:2003

ISO/IEC TR 9126-2:2003 provides external metrics for measuring attributes of six external quality characteristics defined in ISO/IEC 9126-1. ISO/IEC TR 9126-2:2003 defines external metrics, ISO/IEC TR 9126-3 defines internal metrics and ISO/IEC 9126-4 defines quality in use metrics, for measurement of the characteristics or the subcharacteristics. Internal metrics measure the software itself, external metrics measure the behaviour of the computer-based system that includes the software, and quality in use metrics measure the effects of using the software in a specific context of use.

The metrics listed in ISO/IEC TR 9126-2:2003 are not intended to be an exhaustive set. Developers, evaluators, quality managers and acquirers may select metrics from ISO/IEC TR 9126-2:2003 for defining requirements, evaluating software products, measuring quality aspects and other purposes.

Users of ISO/IEC TR 9126-2:2003 can select or modify and apply metrics and measures from ISO/IEC TR 9126-2:2003 or may define application-specific metrics for their individual application domain.

ISO/IEC TR 9126-2:2003 is intended to be used together with ISO/IEC 9126-1.

ISO/IEC TR 9126-2:2003 contains an explanation of how to apply software quality metrics, a basic set of metrics for each subcharacteristic and an example of how to apply metrics during the software product life cycle.

ISO/IEC TR 9126-2:2003 does not assign ranges of values of these metrics to rated levels or to grades of compliance, because these values are defined for each software product or a part of the software product, by its nature, depending on such factors as category of the software, integrity level and users' needs. Some attributes may have a desirable range of values, which does not depend on specific user needs but depends on generic factors; for example, human cognitive factors.

ISO/IEC TR 9126-3:2003

ISO/IEC TR 9126-3:2003 provides internal metrics for measuring attributes of six external quality characteristics defined in ISO/IEC 9126-1. ISO/IEC TR 9126-2 defines external metrics, ISO/IEC TR 9126-3:2003 defines internal metrics and ISO/IEC 9126-4 defines quality in use metrics, for measurement of the characteristics or the subcharacteristics. Internal metrics measure the software itself, external metrics measure the behaviour of the computer-based system that includes the software, and quality in use metrics measure the effects of using the software in a specific context of use.

The metrics listed in ISO/IEC TR 9126-3:2003 are not intended to be an exhaustive set. Developers, evaluators, quality managers, maintainers, suppliers, users and acquirers may select metrics from ISO/IEC TR 9126-3:2003 for defining requirements, evaluating software products, measuring quality aspects and other purposes.

Users of ISO/IEC TR 9126-3:2003 can select or modify and apply metrics and measures from ISO/IEC TR 9126-3:2003 or may define application-specific metrics for their individual application domain. For internal metrics view, there are pure internal metrics proposed for reference purposes.

ISO/IEC TR 9126-3:2003 is intended to be used together with ISO/IEC 9126-1.

ISO/IEC TR 9126-3:2003 contains:

- an explanation of how to apply software quality metrics;
- a basic set of metrics for each subcharacteristic;

- an example of how to apply metrics during the software product life cycle.

ISO/IEC TR 9126-3:2003 does not assign ranges of values of these metrics to rated levels or to grades of compliance, because these values are defined for each software product or a part of the software product, by its nature, depending on such factors as category of the software, integrity level and users' needs. Some attributes may have a desirable range of values, which does not depend on specific user needs but depends on generic factors; for example, human cognitive factors.

ISO/IEC TR 9126-4:2004

ISO/IEC TR 9126-4:2004 provides quality in use metrics for measuring the attributes defined in ISO/IEC 9126-1. ISO/IEC TR 9126-2 defines external metrics and ISO/IEC TR 9126-3 defines internal metrics for measurement of the subcharacteristics defined in ISO/IEC 9126-1. Internal metrics measure the software itself, external metrics measure the behaviour of the computer-based system that includes the software, and quality in use metrics measure the effects of using the software in a specific context of use.

The metrics listed in ISO/IEC TR 9126-4 are not intended to be an exhaustive set. Developers, evaluators, quality managers and acquirers may select metrics from ISO/IEC TR 9126-4 for defining requirements, evaluating software products, measuring quality aspects and other purposes.

ISO/IEC TR 9126-2 is intended to be used together with ISO/IEC 9126-1.

ISO/IEC TR 9126-4 contains:

-- an explanation of how to apply software quality metrics;

-- a basic set of metrics for each characteristic; and

-- an example of how to apply metrics during the software product life cycle.

It includes as informative annexes a quality in use evaluation process and a reporting format

ISO/IEC 25000:2005

ISO/IEC 25000:2005 provides guidance for the use of the new series of International Standards named Software product Quality Requirements and Evaluation (SQuaRE). The purpose of this guide is to provide a general overview of SQuaRE contents, common reference models and definitions, as well as the relationship among the documents, allowing users of this guide a good understanding of those series of International Standards, according to their purpose of use. This document contains an explanation of the transition process between the old ISO/IEC 9126 and the 14598 series and SQuaRE, and also presents information on how to use the ISO/IEC 9126 and 14598 series in their previous form.

SQuaRE provides:

- Terms and definitions,
- Reference models,
- General guide,
- Individual division guides, and
- Standards for requirements specification, planning and management, measurement and evaluation purposes.

Software development is, quite possibly, one of the fastest moving, and changing, areas within the field encompassed by the term engineering. Accordingly, the latest information on changes to models and methodologies should always be sought before commencement of a project. The most appropriate and cost effective solutions must be decided on, and implemented, from the onset of the project and utilised throughout the full life cycle of the project, allowing for legislation it is very unlikely that any changes to the chosen path, taken further down the line, will result in cost benefits for the project.

LEAFLET 52/0

BIBLIOGRAPHY

Standards

1. BS 5760 Part 8, "Reliability of systems, equipment and components. Guide to assessment of reliability of systems containing software", British Standards Institute, October 1998.
2. DEF STAN 00-42 part 4, "R&M Assurance Guide – Testability", Issue 1, MoD, 13th December 2002.
3. IEC 62628, Guidance on Software Aspects of Dependability.

Papers

4. "Rigorously Assessing Software Reliability and Safety", Strigini and Fenton, ESA Software Product Assurance Workshop, Nordvjik, March 1996.
5. "Validation of Ultra High Dependability for Software-based Systems", Littlewood and Strigini, ACM, 1993.
6. "The Facts About Predicting Software Defects and Reliability", Neufelder-Owner, A., N., Journal of the RAC, 2ndQ, 2002, pp 1-4.
7. "Probability and Statistics with Reliability, Queuing, and Computer Science Applications", Trivedi, Kishor S, John Wiley and Sons, New York, 2001, ISBN number 0-471-33341-7.
8. "Stopping rules for the operational testing of safety-critical software", Littlewood B. & Wright D., Digest of IEEE 1995 FTC'S, *25th Annual International Symposium Fault-Tolerant Computing*, (Pasadena), IEEE Computer Society Silver Spring, Md., pp444-451, 1995.